Expressiveness via Leader Election Problems

Maria Grazia Vigliotti, Iain Phillips, Catuscia Palamidessi

Separation results via leader election problems

Maria Grazia Vigliotti[1,3], Iain Phillips[2], and Catuscia Palamidessi[3,4]

[1] INRIA Sophia-Antipolis, France
[2] Department of Computing, Imperial College London, England
[3] INRIA Futurs, France
[4] LIX Polytechnique, France

Abstract. We compare the expressive power of process calculi by studying the problem of electing a leader in a symmetric network of processes. We consider the π-calculus with mixed choice and with separate choice, value-passing CCS and Mobile Ambients. We provide a unified approach for all these calculi using reduction semantics.

1 Introduction

Expressiveness results, in formal languages, deal primarily with the question of the power of the underpinning formal model. Different models of computation can be compared using the notion of *encoding*. A typical example comes from classical computability theory: Turing machines, Unlimited Register Machines and Lambda Calculus are considered to be *equally powerful* since they can be reciprocally *encoded*.

In this tutorial we consider expressiveness results about concurrent models of computation. In the last twenty years, many different concurrent calculi have been developed, and most of them are Turing complete, that is, they can compute the same functions as Turing machines. However, function computability is only one possible way to evaluate the power of a concurrent language: other aspects, related to the concurrent nature of the model, should also be taken into account. Our focus is on the synchronisation capabilities of a calculus, and more precisely on the mechanisms that allow remote processes to achieve an agreement. Agreement is considered an important problem in Distributed Computing and a lot of research has been devoted to finding algorithms to achieve it, or, depending on the model of computation, proving its impossibility. Apart from the theoretical interest, the problem has important implications of a practical nature in the field of Distributed Systems, where the design of the operating system has to ensure the correct interaction between remote processes in a distributed setting, when a central coordinator is not available or not feasible. Also, in the implementation of (concurrent) languages one has to face the problem of whether certain interaction primitives are too expressive to be implemented in a given distributed architecture.

One approach to comparing two calculi is to exhibit an encoding or to show that such an encoding cannot exist. The notion of encoding is, naturally, subject to specific conditions. For instance, the encoding should not itself solve the

problem of synchronisation: it would be like mapping Turing machines into finite automata by using a translation which adds an oracle.

To show that an encoding does not exist, one way of proceeding is to show that there is a problem that can be solved in one calculus, but not in the other. In the field of distributed algorithms [11, 30], various models of computation have been compared via the *symmetric leader election problem*, which consists in requiring the members of a symmetric network to elect one of them as their leader. The difficulty consists in breaking the initial symmetry to achieve a situation which is inherently asymmetric (one is the leader and the others are not). This method has proved rather successful also for the comparison of various process calculi [3, 20, 7, 22, 21, 24, 23, 25, 31]. In the case of process calculi, actually, some of the symmetry-breaking arguments are rather sophisticated and use additional discriminations that are related to the topology of the network. In other words, some calculi admit a solution to leader election problems only if the network has a specific topology, such as a fully connected graph.

In this tutorial we shall collect, present, systematise and interpret a collection of results regarding expressiveness in process calculi obtained by means of the *symmetric leader election problem*. We shall provide a uniform presentation by the use of reduction semantics, and we shall highlight the similarities and differences between the various approaches to leader election problems. In particular, we shall focus on the following calculi: Communicating Concurrent Systems (CCS), the π-calculus with mixed choice (π_{m}) and with separate choice (π_{s}), and Mobile Ambients (MA).

CCS [12, 14] is a simple calculus, that aims to represent concurrency with synchronous communication. Based on the concept of channels, it contains two primitives for sending and receiving which can synchronise by handshaking on the same channel. In this paper we shall consider *value-passing* CCS, where input and output primitives carry value parameters. However, for the sake of simplicity, we shall call it CCS throughout the paper.

The π-calculus [15] enhances the CCS model by allowing processes to communicate channel names, which can also be used as channels for communication, allowing the dynamic creation of new links between processes (link mobility). In this paper we do not consider the full π-calculus as originally presented; we omit the matching operator and require choices to be guarded, as in [14]. We call this version the *mixed-choice* π-calculus, which we denote by π_{m}; here the word "mixed" signifies that a choice can contain both input and output guards. CCS as we shall present it can be seen as a subset of π_{m}.

The asynchronous π-calculus [10, 2] has become particularly popular as a model for asynchronous communication. In this fragment there is no explicit choice, and outputs have no continuation. However output prefixing and separate choice can be encoded in the asynchronous π-calculus [2, 19]; separate choice is guarded choice with the restriction that input and output guards cannot be mixed in the same choice. In this tutorial we look at the *separate-choice* π-calculus, which we denote by π_{s}, rather than the asynchronous π-calculus; however the results valid for π_{s} also hold for the asynchronous π-calculus.

Finally, we shall deal with Mobile Ambients. MA [5] has been proposed to model features of computation over the Internet. This calculus is based on the simple unifying concept of *ambient*. Computation is no longer defined as exchanging values, but it is the result of ambients moving into and out of other ambients bringing along active processes and possibly other ambients.

Several relations among the above calculi are obvious or have been proved in the literature, addressing at least partially the issue of expressiveness. However, questions about their expressive power can still be asked:

- π_{s} is a subcalculus of π_{m}. *Is* π_{m} *strictly more expressive?*
- CCS with value passing can be viewed as a subcalculus of π_{m}. Thus π_{m} is as least as expressive as CCS. *Does an encoding exist from* π_{m} *into CCS?*
- The asynchronous π-calculus can be encoded into MA. *Can MA be encoded into the asynchronous* π*-calculus or CCS*?

In the tutorial we shall show that the answers to the previous questions are negative, i.e. those encodings do not exist under certain conditions (Section 2.3). The proofs are based on the possibility/impossibility of solving the symmetric leader election problem.

In encodings of languages that (do not) admit a solution for leader election problems, one important requirement is that the encoding preserves the original distribution among processes. This requirement aims at avoiding that the encoding may introduce a central coordinator [21, 23]. Therefore this condition makes the notion of encoding suitable to compare expressiveness of languages for distributed systems, where processes are expected to coordinate without the help of a centralised server.

The negative results mentioned above have been achieved in recent years as follows:

- Palamidessi [20, 21] established that π_{m} is strictly more expressive than π_{s};
- Phillips and Vigliotti [24, 23, 31] proved a that small fragment of MA is not encodable in π_{s}.

Both those separation results are proved by considering the leader election problem in a fully connected (and symmetric) network. For instance, Palamidessi showed that the problem can be solved in the case of π_{m}, but not in the case of π_{s}. If there were an encoding from π_{m} to π_{s}, then the solution for π_{m} could be translated into one for π_{s}, provided that the encoding satisfied certain conditions (such as distribution—see Section 2.3). No such encoding can exist.

Moreover, finer-grained separation results are proved by considering the leader election problem in a network whose underlying graph is a ring. Those latter negative results have been achieved in recent years as follows:

- Palamidessi [20, 21] proved that CCS does not admit a solution to the leader election problem for certain symmetric rings, while π_{m} does. She deduced that there is no encoding from π_{m} into CCS.
- Phillips and Vigliotti [25] proved that a subcalculus of MA admits a solution to the leader election problem for symmetric rings. They concluded that this calculus cannot be encoded into CCS.

The tutorial is organised in three parts as follows: (1) A general part where we discuss leader election in distributed networks, and how to formalise the problem in process calculi (Section 2). In Section 3 we define the various calculi we shall consider. (2) A part where we deal with leader election problems in general symmetric networks (with no restriction on topology) (Section 4). We present solutions for various calculi, show that other calculi do not admit solutions, and derive separation results. (3) A part where we deal with leader election problems in rings (Section 5). We shall present positive and negative results for various calculi, and again derive separation results. We end the tutorial with a history of related work and conclusions.

2 Leader Election, Electoral Systems and Encodings

After first discussing leader election informally, we show how it can be formalised in the setting of process calculi and reduction semantics. We then discuss criteria for encodings between calculi.

2.1 Leader Election Problems in Distributed Systems

In this section we introduce leader election problems as described in the field of distributed systems. We talk of problems in the plural, because there are different settings that lead to diverse solutions (when solutions do exist). A network is informally a set of machines that run independently and that compute through communication. Abstractly we can think of them as processes. Processes have the same state, if they can perform intuitively the same actions. The essence of a symmetric leader election problem is to find an algorithm where, starting from a configuration (network) of processes in the *same state*, any possible computation reaches a configuration where *one* process is in the state of *leader* and the other processes are in the state *lost* (i.e. they have lost the election). In some cases a solution may be impossible, and in other cases there may be more than one algorithm, and then complexity measures can be used in order to compare the different solutions. In this tutorial, we shall not consider such issues.

The criteria common to all leader election problems are the following:

Symmetry Each process in the network has to have the *same duties*. This is a necessary requirement in order not to trivialise the problem. In fact, in an asymmetric configuration of processes, one process can declare itself the winner. This is not possible in symmetric configurations, since if one process can declare itself the winner, every other process in the configuration can do the same. Thus, in symmetric networks, for the winner to be elected, the *initial symmetry* has to be somehow broken.

Distribution The computation has to be *decentralised*, in the sense that the computation has to start from any subset of processes in the network or configuration. In general, leader election problems are run after a reconfiguration or crash of a system, to the end of establishing which process can start the

initialisation. In this context, the configuration of processes has to be able to elect a leader without any help from outside.

Uniqueness of the leader The processes in a network reach a *final configuration* from *any* computation. In the final configuration there is *one process only* that is elected the *winner* and the other processes in the configuration have lost.

Leader election problems may vary according to the following parameters:

Topology of the network The network could be a *fully connected graph* or a *ring* or *tree* or any other graph or hyper-graph [1, 30, 11]. The topology of the network influences the construction of the algorithm, since it changes the information regarding the totality of the processes involved.
In this tutorial we look at general networks, where there is no restriction on topology, in Section 4, and at rings in Section 5. In the general case, our algorithms will assume that the network is fully connected, though of course this is not assumed when we state impossibility results.

Knowledge of size of the network The number of processes can be known or unknown to the processes before starting the election [30]. This parameter also influences the construction of an algorithm. In this tutorial we shall implement algorithms where the size of the network is known.

Declaration of the leader The leader could be announced by one process only, which could be the leader itself or any other process. Alternatively every process in the configuration has to be aware of the winner. The latter requirement is considered standard, although the weaker one (the former one) is also acceptable, since the winner could inform the other processes of the outcome of the election.
We shall adopt the weaker assumption in this tutorial for simplicity. Note that the original paper [21] uses the stronger requirement for her results.

We have described the leader election problem as presented in the field of distributed algorithms. In this field, it is common to reason on what is known as *pseudo-code* [18]. This means that proofs are given by using some form of 'general-enough-language', that is, a mixed ad-hoc Pascal-like language and natural language without any formalised semantics. Nestmann shows that this approach very often hides underpinning problems and assumptions. The formal and rigorous semantics of process algebra, as presented in this tutorial, is therefore an advantage in the description of leader election problems. Formal semantics is necessary when proving that either a given algorithm is the correct solution to a leader election problem, or that no algorithm exists.

2.2 Electoral Systems

In this section we formalise the leader election problem in process calculi using reduction semantics (unlabelled transitions). Milner and Sangiorgi [16] motivated the study of reduction semantics on the grounds that it is a uniform way of describing semantics for calculi that are syntactically different from each

other. In fact, reduction semantics has been widely used for its simplicity and ability to represent uniformly simple process calculi such as CCS [14], first- and second-order name passing-calculi such as the π-calculus and the higher-order π-calculus [16, 28], and more complex calculi such as the Seal Calculus [6] and the Ambient Calculus [5]. Reduction semantics will provide a uniform framework for all calculi we shall consider.

In reduction semantics a process calculus L is identified with: (1) a set of processes; (2) a reduction relation; and (3) an observational predicate. First of all, we assume the existence of a set of names $\mathcal{N}$: the variables $m, n, x, y \ldots$ range over it. Names are meant to be atomic, and they are a useful abstraction to represent objects that in real life we do not want to view as separated, such as identifiers, sequences of bits, etc.

Some operators of a language are *binding*, in the sense that names that fall within their *scope* are called *bound*, and processes that differ in bound variables only are considered identical. Names that are not bound in a process are called *free*. These concepts will be explicitly defined for each concrete syntax considered later in this tutorial.

We assume that a language L contains at least the parallel composition operator $|$ and the restriction operator $\nu n\, P$. We assume that in each calculus $|$ is a fair operator, in the sense that it does not nondeterministically choose the right or the left-hand side process. This semantics will be common to all the calculi we shall consider in this tutorial. Restriction $\nu n\, P$ binds n; it makes the name n private in P. We write $\nu\vec{n}$ instead of $\nu n_1 \ldots \nu n_k$ for some list of names $n_1, \ldots, n_k$ which is not relevant in the context.

The computational steps for a language can be captured by a simple relation over the set of processes called the *reduction relation*, written $\rightarrow$. To model visible behaviour of programs, an *observation relation* is defined between processes and names; $P \downarrow n$ means intuitively that the process P has the observable name n. We shall see in each concrete calculus how these notions are defined.

Networks are informally compositions of processes or processes composed with the operator $|$; the size of a network is the number of processes that can be "regarded as separate units". This means that a composition of processes can be seen as one process only in counting the size of the network. A *symmetric* network is a network where components differ only on their names. Components of a network are connected if they share names, using which they can engage in communication. *Rings* are networks where each process is connected just to its left-hand and right-hand neighbours. A network elects a leader by exhibiting a special name, and an *electoral system* is a network where every possible maximal computation elects a leader.

We now make these notions precise. We assume that $\mathcal{N}$ includes a set of *observables* $\mathsf{Obs} = \{\omega_i : i \in \mathbb{N}\}$, such that for all i, j we have $\omega_i \neq \omega_j$ if $i \neq j$. The observables will be used by networks to communicate with the outside world.

Definition 2.1. *Let P be a process. A* computation *$\mathcal{C}$ of P is a (finite or infinite) sequence $P = P_0 \rightarrow P_1 \rightarrow \cdots$. It is* maximal *if it cannot be extended, i.e. either $\mathcal{C}$ is infinite, or else it is of the form $P_0 \rightarrow \cdots \rightarrow P_h$ where $P_h \not\rightarrow$.*

Definition 2.2. *Let $\mathcal{C}$ be a computation $P_0 \to \cdots \to P_h \to \cdots$. We define the* observables *of $\mathcal{C}$ to be* $\mathsf{Obs}(\mathcal{C}) = \{\omega \in \mathsf{Obs} : \ \exists h \ \ P_h \downarrow \omega\}$.

Networks are collections of processes running in parallel:

Definition 2.3. *A* network Net *of size k is a pair $(A, \langle P_0, \ldots, P_{k-1}\rangle)$, where A is a finite set of names and $P_0, \ldots, P_{k-1}$ are processes. The* process interpretation $\mathsf{Net}^\natural$ *of* Net *is the process $\nu A\,(P_0 \mid \cdots \mid P_{k-1})$. We shall always work up to structural congruence, so that the order in which the restrictions in A are applied is immaterial.*

Networks are to be seen as presentations of processes, showing how the global process is distributed to the k nodes of the network. We shall sometimes write $[P_0 \mid \cdots \mid P_{k-1}]$ instead of $\nu A\,(P_0 \mid \cdots \mid P_{k-1})$, when the globally restricted names do not need to be made explicit.

We shall tend to write networks in their process interpretation (i.e. as restricted parallel compositions), while still making it clear which process belongs to each node of the network.

Networks inherit a notion of computation from processes through the process interpretation: $\mathsf{Net} \to \mathsf{Net}'$ if $\mathsf{Net}^\natural \to \mathsf{Net}'^\natural$. Overloading notation, we shall let $\mathcal{C}$ range over network computations. Also, we define the observables of a network computation $\mathcal{C}$ to be the observables of the corresponding process computation: $\mathsf{Obs}(\mathcal{C}) = \mathsf{Obs}(\mathcal{C}^\natural)$.

The definitions that follow lead up to the formulation of symmetry in a network (Definition 2.7), capturing the notion that each process is the same apart from the renaming of free names.

Definition 2.4. *A* permutation *is a bijection $\sigma : \mathcal{N} \to \mathcal{N}$ such that σ preserves the distinction between observable and non-observable names, i.e. $n \in \mathsf{Obs}$ iff $\sigma(n) \in \mathsf{Obs}$. Any permutation σ gives rise in a standard way to a mapping on processes, where $\sigma(P)$ is the same as P, except that any free name n of P is changed to $\sigma(n)$ in $\sigma(P)$, with bound names being adjusted as necessary to avoid clashes.*

A permutation σ induces a bijection $\hat{\sigma} : \mathbb{N} \to \mathbb{N}$ defined as follows: $\hat{\sigma}(i) = j$ where $\sigma(\omega_i) = \omega_j$. Thus for all $i \in \mathbb{N}$, $\sigma(\omega_i) = \omega_{\hat{\sigma}(i)}$. We use $\hat{\sigma}$ to permute the indices of processes in a network.

Definition 2.5. *Let $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$ be a network of size k. An* automorphism *on* Net *is a permutation σ such that (1) $\hat{\sigma}$ restricted to $\{0, \ldots, k-1\}$ is a bijection, and (2) σ preserves the distinction between free and bound names, i.e. $n \in \vec{n}$ iff $\sigma(n) \in \vec{n}$. If $\hat{\sigma}$ restricted to $\{0, \ldots, k-1\}$ is not the identity we say σ is* non-trivial.

Definition 2.6. *Let σ be an automorphism on a network of size k. For any $i \in \{0, \ldots, k-1\}$ the* orbit *$\mathcal{O}_{\hat{\sigma}}(i)$ generated by $\hat{\sigma}$ is defined as follows:*

$$\mathcal{O}_{\hat{\sigma}}(i) = \{i, \hat{\sigma}(i), \hat{\sigma}^2(i), \ldots, \hat{\sigma}^{h-1}(i)\}$$

where $\hat{\sigma}^j$ represents the composition of $\hat{\sigma}$ with itself j times, and h is least such that $\hat{\sigma}^h(i) = i$. If every orbit has the same size then σ is well-balanced.

Definition 2.7. *Let* $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$ *be a network of size* k *and let* σ *be an automorphism on it. We say that* Net *is* symmetric with respect to σ *iff for each* $i = 0, \ldots, k-1$ *we have* $P_{\hat{\sigma}(i)} = \sigma(P_i)$.

We say that Net *is* symmetric *if it is symmetric with respect to some automorphism with a single orbit (which must have size* k*).*

Intuitively an electoral system is a network which reports a unique winner, no matter how the computation proceeds.

Definition 2.8. *A network* Net *of size* k *is an* electoral system *if for every maximal computation* $\mathcal{C}$ *of* Net *there exists an* $i < k$ *such that* $\mathsf{Obs}(\mathcal{C}) = \{\omega_i\}$.

2.3 Encodings

The concept of encoding is inherently associated to expressiveness. If there exists an encoding $[\![-]\!]$ from a source language S to a target language T, one could see the language T as 'mirroring' S. Thus, the model underpinning S is at least as expressive as the one underpinning T. At the highest level of abstraction, an encoding $[\![-]\!]$ is a function from a source language to a target language. However, not just any function $[\![-]\!]$ from source language to target language should be accepted as an encoding; some 'relevant' behaviour of the first language must be 'preserved'.

We appeal here to the intuitive meaning of the words 'relevant' and 'to preserve', but it remains to formalise the meaning of these words, by exhibiting the semantic properties that $[\![-]\!]$ must satisfy. There is no definitive list of properties that are relevant or that have to be satisfied by an encoding. We shall give below some of the most common ones. Assuming that $P \in S$, and that $\rightarrow^*$ means the reflexive and transitive closure of the reduction relation, we then have:

- Preservation of execution steps (completeness): if $P \rightarrow P'$ then $[\![P]\!] \rightarrow^* [\![P']\!]$ [19, 13, 4];
- Reflection of execution steps (soundness): if $[\![P]\!] \rightarrow^* Q$ then there is P' such that $P \rightarrow^* P'$ and $Q \rightarrow^* [\![P']\!]$ [19, 13];
- Barb preservation (completeness): if $P \downarrow n$ then $[\![P]\!] \downarrow n$ [32];
- Barb reflection (soundness): if $[\![P]\!] \downarrow n$ then $P \downarrow n$ [32].

(Of course, other properties could be added.)

One might also add syntactic requirements on an encoding. To give a concrete example, assuming that $\mid$ and ν are two operators common to S and T, then the statements below express that $[\![-]\!]$ preserves bound names (restriction) and distribution (parallel composition). Clearly the list could be longer, according to the number of common operators in the source and the target language. Other syntactic properties specific to languages could be considered.

- Distribution preservation: $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$ [21, 25, 23];
- Name preservation: $[\![\nu n\, P]\!] = \nu n\, [\![P]\!]$ [7];
- Substitution preservation: for all substitutions σ on S there exists a substitution θ on T such that $[\![\sigma(P)]\!] = \theta([\![P]\!])$ [21, 31];

– Link independence: if $fn(P) \cap fn(Q) = \emptyset$ then $fn([\![P]\!]) \cap fn([\![Q]\!]) = \emptyset$ [21, 25].

(Of course, other properties could be added.)

The list of properties given above is certainly not exhaustive, but it includes some common properties used by the scientific community [19, 5, 21, 23, 25, 32, 8].

In general, it is not required that all of the properties above are satisfied in order for a function to be called an encoding. More specifically, there is not even a subset of these properties that is regarded as *necessary*. In fact, the conditions regarded as relevant depend on the reasons why the encoding is sought in the first place. For instance one could show that some primitives are redundant in a calculus by showing an encoding from the full set of processes to an appropriate fragment. This could be very useful for implementation purposes. This is the case for the programming language Pict [26], which is based on the asynchronous π-calculus, where input-guarded choice can be implemented [19]. One could also show that one calculus can be encoded into another in order to 'inherit' some (possibly good) properties. For instance, from the encoding of the λ-calculus into the π-calculus one could derive easily the Turing completeness of the π-calculus.

In encodings of languages that admit a solution for leader election problems, one important requirement is that the encoding is homomorphic with respect to parallel composition, i.e. preserves distribution. This requirement aims at avoiding that the encoding introduces a trivial solution to such a problem [21, 23]. However, Nestmann [17] and Prasad [27] argue that this requirement is too strong for practical purposes. We would like to defend it, on the basis that it corresponds to requiring that the degree of distribution of the processes is maintained by the translation, i.e. no coordinator is added. This condition makes the notion of encoding suitable to compare expressiveness of languages for distributed systems, where processes are expected to coordinate without the help of a centralised control.

Although there is no unanimous agreement on what constitutes an encoding, it is clear that the judgment as to whether a function is an encoding relies on acceptance or rejection of the properties that hold for the encoding. That is, to give a meaning to the results that will be presented in this tutorial, the conditions on encodings we shall now present have to be accepted and considered 'reasonable'.

In dealing with leader election problems, an encoding must *preserve the fundamental criteria of the problem*, that is, the *conditions* for an encoding must preserve symmetric electoral systems without introducing a solution.

Definition 2.9. *Let L and L' be process languages. An encoding $[\![-]\!] : L \to L'$ is*

1. distribution-preserving *if for all processes P, Q of L, $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$;*
2. permutation-preserving *if for any permutation of names σ in L there exists a permutation θ in L' such that $[\![\sigma(P)]\!] = \theta([\![P]\!])$ and the permutations are compatible on observables, in that for all $i \in \mathbb{N}$ we have $\sigma(\omega_i) = \theta(\omega_i)$, so that $\hat{\sigma}(i) = \hat{\theta}(i)$;*

3. observation-respecting *if for any P in L,*
   (a) *for every maximal computation $\mathcal{C}$ of P there exists a maximal computation $\mathcal{C}'$ of $[\![P]\!]$ such that $\mathsf{Obs}(\mathcal{C}) = \mathsf{Obs}(\mathcal{C}')$;*
   (b) *for every maximal computation $\mathcal{C}$ of $[\![P]\!]$ there exists a maximal computation $\mathcal{C}'$ of P such that $\mathsf{Obs}(\mathcal{C}) = \mathsf{Obs}(\mathcal{C}')$.*

The condition of preserving distribution is important in ruling out encodings which make use of a *central server*. That means that, if the target language does not admit a fully distributed solution to the leader election problem, the encoding cannot introduce a spurious solution. The second condition prevents a trivial solution from being introduced by collapsing all the set of natural numbers $\{0, 1, \ldots, k-1\}$ to a $j \in \mathbb{N}$. The first two items aim to map symmetric networks to symmetric networks of the same size and with the same orbits. The third item aims at preserving the uniqueness of the winner. The condition is on barbs because the winner in this framework is represented with a barb. The conditions of Definition 2.9 have been formulated with the aim of achieving the following lemma, which says that symmetric electoral systems are preserved.

Lemma 2.10. *[24, 23] Let L and L' be process languages. Suppose $[\![-]\!] : L \to L'$ is a uniform observation-respecting encoding. Suppose that Net is a symmetric electoral system of size k in L with no globally bound names. Then $[\![\mathsf{Net}]\!]$ is a symmetric electoral system of size k in L'.* □

3 Calculi

In this section we define the various calculi we shall consider.

3.1 The π-calculus with Mixed Choice

We assume the existence of names $n \in \mathcal{N}$ and co-names $\overline{n} \in \overline{\mathcal{N}}$. The set of process terms of the π-calculus with mixed choice (π_{m}) is given by the following syntax:

$$P, Q ::= \mathbf{0} \mid \textstyle\sum_{i \in I} \alpha_i.P \mid P \mid Q \mid \nu n\, P \mid !\, P$$

where I is a finite set. The *prefixes* of processes, ranged over by α, are defined by the following syntax:

$$\alpha ::= m(n) \mid \overline{m}\langle n \rangle\,.$$

Summation $\sum_{i \in I} \alpha_i.P_i$ represents a finite choice among the different processes $\alpha_i.P$. This operator is also called *mixed choice*, since both input and output prefixes can be present in the same summation. The symbol $\mathbf{0}$, called *nil*, is the inactive process. Commonly in the π-calculus, $\mathbf{0}$ is an abbreviation for the empty choice. Although redundant, we introduce it here as a primitive for uniformity with the syntax of other calculi. *Replication* $!\, P$ simulates recursion by spinning off copies of P. *Parallel composition* of two processes $P \mid Q$ represents P and Q computing independently from each other. *Restriction* $\nu n\, P$ creates a new name n in P, which is bound. We shall feel free to omit trailing $\mathbf{0}$s. Thus we write α

instead of $\alpha.\mathbf{0}$. The notion of the *free names* $fn(P)$ of a term P is standard, taking into account that the only binding operators are input prefix and restriction. We write $P\{n/m\}$ to mean that each free occurrence of m is substituted by n in P. We reserve η for a bijection on I; we write $\sum_{\eta(i)\in I}$ for permutation on the sub-processes in the choice operator. The *reduction relation* over the processes of π_{m} is the smallest relation satisfying the following rules:

$$\text{(Pi Comm)} \quad (m(x).P + G) \mid (\overline{m}\langle n\rangle.Q + H) \to P\{n/x\} \mid Q$$

$$\text{(Par)} \quad \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \text{(Res)} \quad \frac{P \to P'}{\nu n\, P \to \nu n\, P'}$$

$$\text{(Str)} \quad \frac{P \equiv Q \quad Q \to Q' \quad Q' \equiv P'}{P \to P'}$$

where G, H are summations. *Structural congruence* $\equiv$ allows rearrangement of processes; it is the smallest congruence over the set of processes that satisfies the following equations:

$$\begin{array}{rr} P \mid \mathbf{0} \equiv P & \nu n\,(P \mid Q) \equiv P \mid \nu n\, Q \quad \text{if} \quad n \notin fn(P) \\ P \mid Q \equiv Q \mid P & \nu m\, \nu n\, P \equiv \nu n\, \nu m\, P \\ (P \mid Q) \mid R \equiv P \mid (Q \mid R) & !\, P \equiv P \mid !\, P \\ \nu n\, \mathbf{0} \equiv \mathbf{0} & \sum_{i\in I} \alpha_i.P_i \equiv \sum_{\eta(i)\in I} \alpha_{\eta(i)}.P_{\eta(i)} \end{array}$$

together with α-conversion of bound names. A process P *exhibits barb* n, written as $P \downarrow n$, iff $P \equiv \nu\vec{m}\,((\overline{n}\langle x\rangle.Q + G) \mid R)$ with $n \notin \vec{m}$. We only use barbs on outputs; input barbs are not needed, and we thereby obtain greater uniformity across the calculi we are considering.

3.2 The π-calculus with Separate Choice

The π-calculus with separate choice (π_{s}) [29] is the sub-calculus of π_{m} where summations cannot mix input and output guards. The set of processes is given by the following grammar:

$$P, Q ::= \mathbf{0} \mid \sum_{i\in I} \alpha_i^I.P_i \mid \sum_{i\in I} \alpha_i^O.P_i \mid !\, P \mid P|Q \mid \nu n\, P$$

$$\alpha^I ::= m(n) \qquad \alpha^O ::= \overline{m}\langle n\rangle$$

The semantics of this calculus is the same as for π_{m} taking into account the syntactic restrictions. One could regard π_{s} as having the same expressive strength as the asynchronous π-calculus [10, 2], in view of the results on encoding of separate choice [17].

3.3 CCS

In this paper we shall use the version of CCS presented in [14], with the addition of value passing. As well as names $n \in \mathcal{N}$, we use co-names $\overline{n} \in \overline{\mathcal{N}}$, a set $\mathcal{V}$ of

values, ranged over by $v, \ldots$, and a set $\mathcal{W}$ of variables, ranged over by $x, \ldots$. The sets $\mathcal{N}$, $\overline{\mathcal{N}}$, $\mathcal{V}$ and $\mathcal{W}$ are mutually disjoint. Processes are defined as follows:

$$P, Q ::= \mathbf{0} \mid \sum_{i \in I} \pi_i.P_i \mid P \mid Q \mid \nu n\, P \mid A\langle m_1, \ldots, m_k \rangle$$

where I is a finite set. The *prefixes* of processes, ranged over by π, are defined by the following syntax:

$$\pi ::= n(x) \mid \overline{n}\langle v \rangle.$$

Here recursion is handled by process identifiers with parameters; each identifier A is equipped with a defining equation $A(\vec{m}) \stackrel{\text{df}}{=} P_A$. Structural congruence is the same as for π_{m}, except that the law for replication is omitted and we add the following rule for the identifiers:

$$A\langle \vec{n} \rangle \equiv P_A\{\vec{n}/\vec{m}\} \text{ if } A(\vec{m}) \stackrel{\text{df}}{=} P_A \ .$$

The reduction relation has the rule

$$\text{(CCS Comm)} \quad (n(x).P + G) \mid (\overline{n}\langle v \rangle.Q + H) \to P\{v/x\} \mid Q$$

(where G, H are summations) together with (Par), (Res) and (Str) as for π_{m}. The notion of the *free names* $fn(P)$ of a term P is standard, taking into account that the only binding operator on names is restriction. Barbs are much as for π_{m}: a process P *exhibits barb* n, written as $P \downarrow n$, iff $P \equiv \nu \vec{m}\, ((\overline{n}\langle v \rangle.Q + G) \mid R)$ with $n \notin \vec{m}$.

The difference between CCS and π_{m} may be illustrated by the π_{m} process $P \stackrel{\text{df}}{=} a(x).\overline{x}\langle b \rangle$. This is not a valid CCS process, since x cannot be used as a name in CCS. Clearly, when P is composed with $Q \stackrel{\text{df}}{=} \overline{a}\langle c \rangle.Q'$, P can acquire a new name c that may be used for future communication.

3.4 Mobile Ambients

In the presentation of Mobile Ambients, we follow [5], except for communication, as noted below. Let $P, Q, \ldots$ range over processes and $M, \ldots$ over capabilities. We assume a set of names $\mathcal{N}$, ranged over by $m, n, \ldots$. Processes are defined as follows:

$$P, Q ::= \mathbf{0} \mid P \mid Q \mid \nu n\, P \mid !\, P \mid n[\, P \,] \mid M.P \mid (n).P \mid \langle n \rangle$$

We describe here only the operators specific to ambients: $n[\, P \,]$ is an ambient named n containing process P; $M.P$ performs capability M before continuing as P; and $(n).P$ receives input on an anonymous channel, with the input name replacing free occurrences of name n in P; and finally $\langle n \rangle$ is a process which outputs name n. Notice that output is *asynchronous*, that is, it has no continuation. Restriction and input are name-binding, which naturally yields the definition of the free names $fn(P)$ of a given process P.

Capabilities are defined as follows:

$$M ::= \mathsf{in}\, n \mid \mathsf{out}\, n \mid \mathsf{open}\, n$$

Capabilities allow movement of ambients ($\mathsf{in}\, n$ and $\mathsf{out}\, n$) and dissolution of ambients ($\mathsf{open}\, n$).

We confine ourselves in this paper to communication of names, rather than full communication including capabilities (as in [5]). This serves to streamline the presentation; the results would also hold for full communication.

The *reduction* relation $\rightarrow$ is generated by the following rules:

$$\begin{array}{ll}
\text{(In)} & n[\,\mathsf{in}\, m.P \mid Q\,] \mid m[\,R\,] \rightarrow m[\,n[\,P \mid Q\,] \mid R\,] \\
\text{(Out)} & m[\,n[\,\mathsf{out}\, m.P \mid Q\,] \mid R\,] \rightarrow n[\,P \mid Q\,] \mid m[\,R\,] \\
\text{(Open)} & \mathsf{open}\, n.P \mid n[\,Q\,] \rightarrow P \mid Q \\
\text{(MA Comm)} & \langle n \rangle \mid (m).P \rightarrow P\{n/m\} \\
\text{(Amb)} & \dfrac{P \rightarrow P'}{n[\,P\,] \rightarrow n[\,P'\,]}
\end{array}$$

together with rules (Par), (Res) and (Str) as given for π_{m}. Structural congruence is the least congruence generated by the following laws:

$$\begin{array}{lll}
P \mid Q \equiv Q \mid P & \nu n\, \nu m\, P \equiv \nu m\, \nu n\, P & \\
(P \mid Q) \mid R \equiv P \mid (Q \mid R) & \nu n\, (P \mid Q) \equiv P \mid \nu n\, Q & \text{if } n \notin \mathit{fn}(P) \\
P \mid \mathbf{0} \equiv P & \nu n\, m[\,P\,] \equiv m[\,\nu n\, P\,] & \text{if } n \neq m \\
!\,P \equiv P \mid !\,P & \nu n\, \mathbf{0} \equiv \mathbf{0} &
\end{array}$$

together with α-conversion of bound names. The most basic observation we can make of an MA process is the presence of an unrestricted top-level ambient. A process P *exhibits barb* n, written as $P \downarrow n$, iff $P \equiv \nu \vec{m}\, (n[\,Q\,] \mid R)$ with $n \notin \vec{m}$.

4 Leader Election in General Symmetric Networks

We present solutions to the leader election problem for symmetric networks in a variety of calculi (Section 4.1), followed by results showing the impossibility of solutions in other calculi (Section 4.2). We conclude the section by using the preceding to obtain separation results (Section 4.3).

4.1 Calculi with Electoral Systems

In this section we present solutions to the leader election problem in symmetric networks of any finite size in some fragments of CCS, π_{m} and MA. The solutions are of course still valid in the full calculi. The solutions for CCS and π_{m} are the same, since CCS is a subcalculus of π_{m} and therefore once a solution is proposed for CCS it trivially implies that there is a solution for π_{m}.

Definition 4.1. *1. Let $\pi_{\mathsf{m}}^{-\nu}$ be π_{m} but without restriction (public π_{m}).*

2. *Let CCS$^{-\nu}$ be CCS but without restriction (public CCS).*
3. *Let* MA$^{\mathsf{io}}$ *be MA without communication, restriction and the* open *capability (pure public boxed MA).*

We start by defining a symmetric electoral system of size two in CCS$^{-\nu}$. Let a network Net be defined as follows:

$$P_0 \stackrel{\text{df}}{=} x_0(y) + \overline{x_1}\langle z\rangle.\overline{\omega_0}\langle z\rangle \qquad P_1 \stackrel{\text{df}}{=} x_1(y) + \overline{x_0}\langle z\rangle.\overline{\omega_1}\langle z\rangle \qquad \mathsf{Net} \stackrel{\text{df}}{=} P_0 \mid P_1.$$

The network is symmetric with respect to a single-orbit automorphism σ which swaps 1 and 0, with σ the identity on all other names. There are only two possible computations. One is the following:

$$\mathcal{C} : \mathsf{Net} \to \overline{\omega_1}\langle z\rangle \qquad \mathsf{Obs}(\mathcal{C}) = \{\omega_1\}.$$

The other one is identical up to the renaming of σ. The values passed are just dummies, which can be omitted; there is a crucial use of mixed choice to break symmetry.

The previous solution can be generalised to networks of any size k. Before giving the formal definition, we provide an informal description of the algorithm. Successive pairs of processes fight each other. Winning an individual fight is achieved by sending a message to the loser. Each time, the loser drops out of the contest. Eventually only one process is left standing. It has defeated every other process and is therefore the winner. Each node is composed of two parts:

1. A process that either sends a message to another node and proceeds to fight the remaining processes, or receives a message and will no longer take part in the election process. In this latter case, it will announce to every other node that it has lost.
2. A counter, which collects all the messages of loss from the other processes, and after $k-1$ messages declares victory (so processes have to know the size of the network).

One important feature in this implementation is the use of mixed choice, in the description of the process that runs for the election. Let $\prod_{i<k} P_i$ stand for $P_0 \mid \cdots \mid P_{k-1}$.

Theorem 4.2. *For any $k \geq 1$, in CCS$^{-\nu}$ there exists a symmetric electoral system of size k defined by* $\mathsf{Net} \stackrel{\text{df}}{=} \prod_{i<k} P_i$ *where*

$$\begin{aligned}
P_i &\stackrel{\text{df}}{=} \mathit{Elect}_i \mid \mathit{Counter}^k_{i,0} \\
\mathit{Elect}_i &\stackrel{\text{df}}{=} \overline{n_i}.\mathit{Elect}_i + \textstyle\sum_{0\leq s<k, s\neq i} n_s.(\prod_{0\leq t<k, t\neq i} \overline{\mathit{lost}_t}) \\
\mathit{Counter}^k_{i,j} &\stackrel{\text{df}}{=} \mathit{lost}_i.\mathit{Counter}^k_{i,j+1} \quad (0 \leq j < k-1) \\
\mathit{Counter}^k_{i,k-1} &\stackrel{\text{df}}{=} \overline{\omega_i}.
\end{aligned}$$

□

Because CCS$^{-\nu}$ can be regarded as a subcalculus of $\pi^{-\nu}_{\mathsf{m}}$, the algorithm written above is also a solution for $\pi^{-\nu}_{\mathsf{m}}$. Hence:

Corollary 4.3. *For any $k \geq 1$, in $\pi^{-\nu}_{\mathsf{m}}$ there exists a symmetric electoral system of size k.*

We now turn to showing the existence of symmetric electoral systems in MA. In fact we can make do with the fragment $\mathrm{MA}^{\mathsf{io}}$. Before presenting a solution for networks of arbitrary size, we present an electoral system of size two. Let

$$P_0 \stackrel{\text{df}}{=} n_0[\,\mathsf{in}\, n_1.\omega_0[\,\mathsf{out}\, n_0.\mathsf{out}\, n_1\,]\,] \qquad P_1 \stackrel{\text{df}}{=} n_1[\,\mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,]$$
$$\mathsf{Net} \stackrel{\text{df}}{=} P_0 \mid P_1 \ .$$

The network is symmetric with respect to a single-orbit automorphism σ which swaps 1 and 0. There are only two possible computations. We shall present the first one in detail:

$$\begin{aligned}\mathcal{C} : n_0[\,\mathsf{in}\, n_1.\omega_0[\,\mathsf{out}\, n_0.\mathsf{out}\, n_1\,]\,] \mid n_1[\,\mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,] &\rightarrow \\ n_1[\,n_0[\,\omega_0[\,\mathsf{out}\, n_0.\mathsf{out}\, n_1\,]\,] \mid \mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,] &\rightarrow \\ n_1[\,\omega_0[\,\mathsf{out}\, n_1\,] \mid n_0[\,] \mid \mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,] &\rightarrow \\ \omega_0[\,] \mid n_1[\,n_0[\,] \mid \mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,].\end{aligned}$$

Thus we conclude $\mathsf{Obs}(\mathcal{C}) = \{\omega_0\}$. The other computation is identical up to renaming via σ. Notice that symmetry is broken by one ambient entering the other.

The general solution for a network of any size is more complex, and before introducing the technical solution we shall provide an informal description.

The basic idea of the algorithm is that winning the election is achieved by having all the opponents inside. Each process is composed of two ambients: one that runs for the election and the other that has the rôle of a counter. Any ambient entering another one has lost the election. It will release an ambient called *lose*, which will eventually appear at the top level, where the counters are. The winning ambient is left on its own, at the top level, while all the other ambients are inside the winner. The counter will declare the winner once every loser has entered.

Theorem 4.4. *[23] In* $\mathrm{MA}^{\mathsf{io}}$*, for any $k \geq 1$ there exists a symmetric electoral system of size k, defined by* $\mathsf{Net} \stackrel{\text{df}}{=} \prod_{i<k} P_i$ *where*

$$\begin{aligned} P_i &\stackrel{\text{df}}{=} n_i[\,\textstyle\prod_{j\neq i} \mathsf{in}\, n_j.lose_i[\,Outn\,]\,] \mid c_i[\,C_{i,i+1}\,] \\ Outn &\stackrel{\text{df}}{=} \textstyle\prod_{j<k} !\,\mathsf{out}\, n_j \\ C_{i,i} &\stackrel{\text{df}}{=} \omega_i[\,\mathsf{out}\, c_i\,] \\ C_{i,j} &\stackrel{\text{df}}{=} \mathsf{in}\, lose_j.C'_{i,j} \qquad (j \neq i) \\ C'_{i,j} &\stackrel{\text{df}}{=} \mathsf{out}\, lose_j.C_{i,j+1} \qquad (j \neq i) \end{aligned}$$ □

In the preceding theorem we use addition modulo k.

4.2 Calculi without Electoral Systems

In this section we shall show that there are calculi that do not admit a symmetric electoral system. We shall see that certain operators are needed to break symmetry and for a solution to be possible. For π-calculus and CCS the crucial operator is the mixed choice operator. In fact, both π-calculus and CCS with separate choice cannot solve the problem of electing a leader in any graph. In the case of MA, the in capability is the symmetry-breaking operator.

The proof of the impossibility of symmetric leader election has different technical details according to the different formalisms, but there is a common structure. The basic idea is to construct one maximal computation which preserves, at some points, the invariant property of reaching a symmetric state. In fact, in symmetric states, election fails either because no one declares himself the winner or, if anybody declares himself a winner, the other processes in the network can do the same.

To make this more concrete we consider an example in π_{s}.

$$P_0 \stackrel{\text{df}}{=} \overline{n}_0.\omega_0 \mid n_1 \qquad P_1 \stackrel{\text{df}}{=} \overline{n}_1.\omega_1 \mid n_0 \qquad \mathsf{Net} \stackrel{\text{df}}{=} P_0 \mid P_1.$$

The network of size two written above is symmetric with the standard automorphism that swaps 1 and 0, but it is not an electoral system. To see this it is sufficient to follow one maximal computation:

$$\mathcal{C}: \quad P_0 \mid P_1 \rightarrow \omega_0 \mid n_1 \mid \overline{n}_1.\omega_1 \rightarrow \omega_0 \mid \omega_1 \ .$$

This example shows that, after the initial step breaking symmetry made by P_0 in trying to declare himself the winner, P_1 can respond in a similar way, which leads to a symmetric network again. Finally, no leader is elected because there is more than one winner: $\mathsf{Obs}(\mathcal{C}) = \{\omega_1, \omega_0\}$. The proof for the general case follows closely such reasoning; each time a step is made by a process (or pair of processes), the other processes can mimic this step, in such a way that symmetry is reached again, and no winner is possible.

There is no solution to the leader election problem in π_{s}:

Theorem 4.5. *[21] Let* $\mathsf{Net} = [P_0 \mid \cdots \mid P_{k-1}]$ *with* $k \geq 2$ *be a symmetric network in* π_{s}*. Then* Net *cannot be an electoral system.* □

A similar theorem could be stated for CCS with separate choice; however, unlike π_{s}, such a calculus has never been considered, and therefore we leave out the statement. It is clear that the mixed choice operator is the key for the expressiveness result in the π-calculus. In MA, the in capability is crucial in order to break the symmetry; in fact, if this is removed, the leader election problem cannot be solved in any graph.

Definition 4.6. *Let* $\mathrm{MA}^{-\mathsf{in}}$ *denote MA without the* in *capability.*

Theorem 4.7. *[24, 23] Let* $\mathsf{Net} = [P_0 \mid \cdots \mid P_{k-1}]$ *with* $k \geq 2$ *be a symmetric network in* $\mathrm{MA}^{-\mathsf{in}}$*. Then* Net *cannot be an electoral system.* □

4.3 Separation Results

By Lemma 2.10, a uniform observation-respecting encoding maps symmetric electoral systems (with no globally bound names) to symmetric electoral systems. So for instance we can now deduce that there can be no uniform observation-respecting encoding from π_{m} into π_{s}, since the former has a symmetric electoral system of at least size two (from Corollary 4.3) and the latter does not (Theorem 4.5).

We can tabulate the positive results of Section 4.1 and the negative results of Section 4.2 in the following diagram:

$$\begin{array}{ccc} \mathrm{CCS}^{-\nu} & \pi_{\mathsf{m}}^{-\nu} & \mathrm{MA}^{\mathsf{io}} \\ \hline & \pi_{\mathsf{s}} \quad \mathrm{MA}^{-\mathsf{in}} & \end{array}$$

All calculi above the line have symmetric electoral systems for any finite size. Those below the line do not have symmetric electoral systems for any size greater than one. Therefore there is no uniform, observation-respecting encoding from any calculus above the line to any below the line, giving us many separation results.

5 Leader Election in Symmetric Rings

In distributed computing, one standard network topology is a ring, where each process can only communicate with its left-hand and right-hand neighbours. As far as leader election is concerned, this means that algorithms which assume that all processes are directly linked to all other processes (as considered in Section 4) will no longer work. In this section we examine whether enhanced leader election algorithms which can handle rings are available for the languages we are considering. This will enable us to separate some of the languages in the top row of the diagram in Section 4.3.

One possible way to conduct leader election in rings is what we shall call the *two-phase* method. This starts by using an algorithm to create links between all processes. Symmetry is preserved during this first (or *link-creation*) phase. Once this is done, in the second (or *election*) phase a leader election algorithm devised for fully connected networks (as in Section 4) can be used to produce the leader.

The π-calculus has the power to create new links; we shall see that the link-creation phase referred to above can be carried out in π_{m} (in fact it can be done in π_{s}). Since π_{m} can solve leader election for fully connected networks, it can therefore perform leader election on rings using the two-phase method. By contrast, CCS does not have the power to create new links; therefore CCS cannot perform leader election on rings with composite (non-prime) size.

We now consider the ambient world. In MA, the communication primitives have the same operational semantics as the π-calculus, except that they are *anonymous*, in the sense that there are no channels on which communication happens (in the π-calculus one would write $m(x).P$ for an input on the channel

m, while in MA one would write $(x).P$ for an anonymous input). Thus, since the communication primitives in ambients are very similar to those of the π-calculus, it would be not surprising if the two-phase method could be formulated in MA, since MA can solve the leader election problem in fully connected networks. However, the leader election problem for symmetric rings of any size is solved without the use of communication primitives. This means that link passing, in this case, is somehow simulated, since there is no explicit way of passing names in the absence of communication. The open capability is crucial in this setting. It is, in fact, the capability that simulates link passing, since it can be shown that MA, without the open capability does not admit a solution for leader election problems in rings of composite size.

5.1 Rings and Independence Preservation

We start by providing a general framework for leader election problems in rings, augmenting that presented in Section 2.2. Note that in our framework, unlike in the standard distributed systems literature, we do not distinguish between unidirectional rings, where messages are passed in one direction only, say from left-hand to right-hand neighbours or vice-versa, and bidirectional rings, where communication can flow in either direction.

Given a network $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$, we can associate a graph with Net by letting the set of nodes be $\{0, \ldots, k-1\}$ and letting $i, j < k$ be adjacent iff $fn(P_i) \cap fn(P_j) \neq \emptyset$. A network forms a ring if the processes can be arranged in a cycle, and each node i is adjacent to at most its two neighbours in the cycle.

Definition 5.1. *A* ring *is a network* $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$ *which has a single-orbit automorphism* σ *such that for all* $i, j < k$*, if* $fn(P_i) \cap fn(P_j) \neq \emptyset$ *then one of* $i = j$*,* $\hat{\sigma}(i) = j$ *or* $\hat{\sigma}(j) = i$ *must hold. A ring is* symmetric *if it is symmetric with respect to such an automorphism* σ*.*

Notice that the definition bans links between non-adjacent nodes in the ring, but does not require the existence of links between adjacent nodes. Thus a completely disconnected network is a ring.

Recall that an *independent set* in a graph is a set of nodes such that no two nodes of the set are adjacent.

Definition 5.2. *Two processes* P *and* Q *are* independent *if they do not share any free names:* $fn(P) \cap fn(Q) = \emptyset$*.*

Definition 5.3. *Let* σ *be an automorphism on a network* $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$*. Then* Net *is* independent *with respect to* σ *if every orbit forms an independent set, in the sense that if* $i, j < k$ *are in the same orbit of* $\hat{\sigma}$ *with* $i \neq j$*, then* P_i *and* P_j *are independent.*

Unlike in Section 4, in this section we shall consider encodings which map rings to rings. We therefore need a further property on top of uniformity and the preservation of the observables. This property will guarantee that the connectivity of the original network is not increased.

Definition 5.4. *An encoding is* independence-preserving *if for any processes P, Q, if P and Q are independent then $[\![P]\!]$ and $[\![Q]\!]$ are also independent.*

The property above states that such an encoding "does not increase the level of connectivity of the network".

Lemma 5.5. *[25] Suppose $[\![-]\!] : L \to L'$ is a uniform, observation-respecting and independence-preserving encoding. Suppose that* Net *is a symmetric ring of size $k \geq 1$ which is an electoral system. Then $[\![\mathsf{Net}]\!]$ is also a symmetric ring of size k which is an electoral system.* □

5.2 Calculi with Electoral Systems for Rings

In this section we show that we can solve leader election on symmetric rings in π_{m} and in MA. We start with a solution to the leader election problem for rings in π_{m}. The algorithm has two phases. In phase one the processes pass names around the ring so that every process becomes directly connected to every other process. Here there is an essential use of the π-calculus, though without any use of choice.

We define a symmetric ring $P_0 \mid \cdots \mid P_{k-1}$ which is an electoral system. Suppose that process P_i has a channel n_i initially known only to itself, and can send messages to P_{i-1} along channel x_i. Then the names n_i are passed around the ring so that all processes share them and can use them in the election phase. We have to be careful that for each P_i the outputs occur in the same order as the inputs, so that names do not get confused. We therefore allocate to each P_i a "synchroniser" name y_i which ensures that each successive output is completed before the next one is enabled. We elide the dummy names passed along y_i.

For $0 \leq i \leq k$, we let $P_i \stackrel{\text{df}}{=} P_i^0\langle x_i, x_{i+1}, y_i, n_i\rangle$, where for $0 \leq j \leq k-2$ we let

$$P_i^j(x_i, x_{i+1}, y_i, n_i, \ldots, n_{i+j}) \stackrel{\text{df}}{=} \bar{x}_i\langle n_{i+j}\rangle.\bar{y_i} \mid x_{i+1}(n_{i+j+1}).y_i.P_i^{j+1}\langle x_i, x_{i+1}, y_i, n_i, \ldots, n_{i+j+1}\rangle$$

and $P_i^{k-1}(x_i, x_{i+1}, y_i, n_i, \ldots, n_{i-1}) \stackrel{\text{df}}{=} Q_i\langle n_i, \ldots, n_{i-1}\rangle$. Here Q_i is a process which has acquired all the n_i and is ready to carry out the election phase. Once Q_i is reached, the names x_i, x_{i+1} and y_i are no longer required.

For π_{m}, we have seen what the Q_i would look like in Theorem 4.2, and therefore we can state the following theorem:

Theorem 5.6. *(cf. [21]) For any $k \geq 1$, there is a symmetric ring of size k which is an electoral system in $\pi_{\mathsf{m}}^{-\nu}$.* □

We now discuss the solution to the leader election problem for rings in pure public MA (i.e. MA without communication and restriction). We use the two-phase method. In the link-creation phase we send ambients round the ring which contain the appropriate capabilities. These are opened by their intended recipients, which then can exercise these capabilities. We already know how to carry out the election phase from Theorem 4.4, though in fact we use a different algorithm,

which is easier to set up via the link-creation phase. We omit the precise details of the construction, as they are quite lengthy.

Theorem 5.7. *[25] For any $k \geq 1$ there is a symmetric ring of size k which is an electoral system in pure public MA.* □

5.3 Calculi without Electoral Systems for Rings

In this section, we consider the calculi that do not have electoral systems for symmetric rings. In this case, the failure of the election is not related to the ability of breaking the initial symmetry. In fact in CCS or MA^{io}, leader election problems can be solved in fully connected networks. The separation results say something regarding the possibility of creating new shared resources. In the π-calculus this phenomenon is present since channels can be values as well; in MA, this phenomenon is simulated via the open capability. Thus, CCS and boxed MA (i.e. MA without the open capability) do not admit a solution to the leader election problem in rings.

As in the case of general networks, the proofs for the negative results differ in their technical details in each formalism, but there is a common strategy. If a ring is of composite (non-prime) size, then it is symmetric with respect to a permutation with multiple independent orbits of the same size. The basic idea is to show that there is a maximal computation where, even though symmetry may be broken in the ring as a whole, symmetry is maintained within each orbit, and the nodes of each orbit remain independent. It remains an open problem whether the result presented below still holds in networks whose size is a prime number.

Theorem 5.8. *[21, 25] For any composite $k > 1$, CCS and boxed MA do not have a symmetric ring of size k which is an electoral system.* □

5.4 Separation Results

By Lemma 5.5, we can now deduce that there can be no uniform, observation-respecting and independence-preserving encoding from π_{m} into CCS, since the former has a symmetric electoral system which is a ring of size four (from Theorem 5.6) and the latter does not (Theorem 5.8).

Much as in Section 4.3, we can tabulate the results of Sections 5.2 and 5.3 as follows:

$\pi_{\mathsf{m}}^{-\nu}$	pure public MA
CCS	boxed MA

All calculi above the line have symmetric electoral systems which are rings for any finite size. Those below the line do not have symmetric electoral systems which are rings for composite sizes greater than one. Therefore there is no uniform, observation-respecting and independence-preserving encoding from either calculus above the line to either below the line.

6 Conclusions and Related Work

The first attempt to represent leader election problems in process algebra was made by Bougé [3]. He formalised the notion of leader election problem in symmetric networks for CSP [9]. The most remarkable achievements are the separation results between CSP with input and output guards and CSP with input guards only, and between the latter and CSP without guards, based on the notion of *symmetric reasonable implementation*.

A similar formalisation of the notion of leader election problem was made by Palamidessi [21] for the π-calculus. Palamidessi proves *formally* that any symmetric network in the π-calculus with separate choice admits a computation that never breaks the initial symmetry. This result is used to show that there is no encoding of the π-calculus with mixed choice into the π-calculus with separate choice. In her paper Palamidessi uses a graph framework, as in the tradition of distributed algorithms [11, 30, 1, 3], and she proves that CCS [12] does not admit a symmetric electoral system in a ring, as opposed to the π-calculus with mixed choice. Using a similar approach Ene and Muntean [7] show that the π-calculus with broadcasting primitives cannot be encoded in the standard π-calculus.

Finally, Phillips and Vigliotti used these proof techniques to separate MA from the separate-choice π-calculus and $\mathrm{MA}^{-\mathsf{in}}$ [23], and mixed choice π-calculus and MA from CCS and boxed MA [25]. This work was carried out in the reduction semantics framework used also in this tutorial. This framework has the advantage of uniformity across a range of process calculi. Our results say nothing, with respect to leader election, on the relationship between the mixed choice π-calculus and MA, or between CCS and boxed MA. These are still open problems.

In this tutorial we have collected together results from different papers [20, 21, 24, 23, 25], given a uniform presentation and highlighted the similarities and differences between the various approaches to leader election problems. We have omitted proofs and lengthy details; however, those are available in the original papers.

7 Acknowledgements

The work of Catuscia Palamidessi and Maria Grazia Vigliotti has been partially supported by the INRIA/ARC project ProNoBiS. Maria Grazia Vigliotti thanks the Group MIMOSA at INRIA Sophia Antipolis for having allowed her to work as guest at their site. We also thank the anonymous referees for their comments.

References

[1] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 82–93. ACM, 1980.

[2] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA Sophia-Antipolis, 1992.

[3] L. Bougé. On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes. *Acta Informatica*, 25:179–201, 1988.

[4] L. Cardelli and A.D. Gordon. Anytime, Anywhere: Modal Logic for Mobile Ambients. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 365–377, 2000.

[5] L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[6] G. Castagna, J. Vitek, and F. Zappa Nardelli. The Seal calculus. *Information and Computation*, 201(1):1–54, 2005.

[7] C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In *Proceedings of 12th International Symposium on Fundamentals of Computation Theory (FCT'99)*, volume 1684 of *Lecture Notes in Computer Science*, pages 258–268. Springer-Verlag, 1999.

[8] D. Gorla. On the relative expressive power of asynchronous communication primitives. In L. Aceto and A. Ingólfsdóttir, editors, *Proceedings of 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 47–62. Springer-Verlag, 2006.

[9] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[10] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP'91)*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer-Verlag, 1991.

[11] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[12] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[13] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):269–310, 1992.

[14] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[15] R. Milner, J. Parrow, and D. Walker. A calculus for mobile processes, parts I and II. *Information and Computation*, 100(1):1–77, 1992.

[16] R. Milner and D. Sangiorgi. Barbed Bisimulation. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[17] U. Nestmann. What is a 'good' encoding of guarded choice? *Information and Computation*, 156:287–319, 2000.

[18] U. Nestmann. Modeling consensus in a process calculus. In *Proceedings of CONCUR'03*, volume 2761 of *Lecture Notes in Computer Science*, pages 393–407. Springer-Verlag, 2003.

[19] U. Nestmann and B.C. Pierce. Decoding Choice Encodings. *Information and Computation*, 163(1):1–59, 2000.

[20] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proceedings of the 25th ACM Symposium on Principles of Programming Languages*, pages 256–265. ACM, 1997.

[21] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[22] I.C.C. Phillips. CCS with priority guards. In *Proceedings of 12th International Conference on Concurrency Theory, CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 305–320. Springer-Verlag, 2001.

[23] I.C.C. Phillips and M.G. Vigliotti. Symmetric electoral systems for ambient calculi. Submitted.

[24] I.C.C. Phillips and M.G. Vigliotti. Electoral systems in ambient calculi. In *Proceedings of 7th International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2004*, volume 2987 of *Lecture Notes in Computer Science*, pages 408–422. Springer-Verlag, 2004.

[25] I.C.C. Phillips and M.G. Vigliotti. Leader election in rings of ambient processes. *Theoretical Computer Science*, 356(3):468–494, 2006.

[26] B.C. Pierce and D.N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 2000.

[27] K.V.S. Prasad. Broadcast Calculus Interpreted in CCS up to Bisimulation. In *Proceedings of Express'01*, volume 52 of *Electronic Notes in Theoretical Computer Science*, pages 83–100. Elsevier, 2002.

[28] D. Sangiorgi. *Expressing Mobility in Process Algebra: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1993.

[29] D. Sangiorgi and D. Walker. *The π-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[30] G. Tel. *Distributed Algorithms*. Cambridge University Press, 2000.

[31] M.G. Vigliotti. *Reduction Semantics for Ambient Calculi*. PhD thesis, Imperial College London, 2004.

[32] N. Yoshida. Graph Types for Monadic Mobile Process Calculi. In *Proceedings of 16th FST/TCS*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer-Verlag, 1996.